The Role of *Custom* Design in ASIC Chips

William J. Dally and Andrew Chang
Computer Systems Laboratory
Stanford University
Stanford, CA 94305
{billd,achang}@cva.stanford.edu

Abstract

Custom design, in which the designer controls the physical structure of the chip, can greatly improve the speed, power, and delay of an ASIC chip without affecting design time. Through floorplanning and tiling data paths, the designer places the critical wires first, before the logic is placed. *Crafted* datapath cells structure wiring at the other end of the spectrum by keeping local wires short enabling the use of minimum sized drivers. Routing the wires first gives early visibility of timing issues, allows the design to be optimized to drive the exact wire load, and enables the use of fast circuit styles.

1 Introduction

We have found that the performance of an ASIC can be greatly improved without increasing design time by judiciously employing a number of *custom* design techniques including floorplanning, prerouting critical signals, tiling datapaths, and generating*crafted cells*. These techniques all structure the design by routing the critical wires first and then placing the devices. This 'key wires first' approach exploits the structure of the logic to reduce wire loads, provide early visibility of the timing and power dissipation of a design, and gives the designer control of the key wiring. These techniques can all be applied within a standard ASIC CAD flow alongside less critical blocks that are implemented without structure.

There is a continuum of design styles between full custom, where every wire is hand placed, to fully automated, where none are. The key to applying custom design techniques to an ASIC is to limit customization by structuring the few key wires that give the most leverage and leaving the rest to automated tools. Examples of critical wires that are easy to structure are global signals and buses, datapath bit and word lines, and signals internal to crafted cells.

Circuits play an important, but subordinate, role in custom design. Structured wiring is the key enabler for fast circuits such as domino logic and low-swing signaling. These circuits require a well controlled load and noise environment that cannot be guaranteed with automated wiring. When the wiring is well controlled the use of fast circuits gives an additional performance boost, but one that is less than a factor of 2.

Traditionally, custom design has been restricted to high-performance components like microprocessors[3]. However, as geometries continue to shrink, both the importance of wire delay and the increase in ASIC complexity will necessitate a significant expansion in the use of custom techniques as automated flows increasingly fail to meet performance, power, and area goals.

2 Custom design means designer control over the physical structure

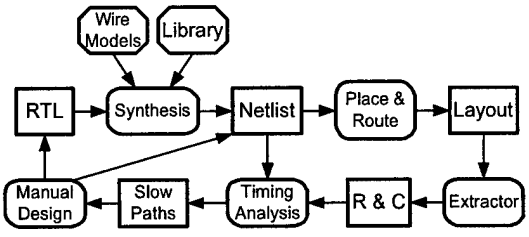
A *custom* design is one in which the designer has explicit control over the physical structure of the design. This is in contrast to *automated* design in which the designer specifies the logical structure of the design and the physical design is generated automatically, usually with little structure. Custom design gives much better circuits because it results in structured wiring where key signals have much smaller wire loads.

In a custom design one first routes the key signals and then places the modules. In contrast, an automated design system places the modules and then routes the signals. This loses the structure of the design. Any EE student knows to layout a datapath in an array of bits and functions; however, no place and route system we know of can automatically perform this task. Data paths are not a special case. Almost every piece of logic we've worked with has a structure that is known to the designer and lost to the tools.

There is a continuum of custom design approaches from simple floorplanning to hand-crafting every transistor. The biggest gains in power and performance come from taking the first few steps down this path. Almost everyone who builds ASICs today takes at least the first step by floorplanning the chip.

2.1 Standard cell design converts RTL to layout without regard for structure

Figure 1: Flow of automated design



A simplified design flow for a standard cell chip is illustrated in Figure 1. The designer writes an RTL description of the function

(Verilog or VHDL). This is synthesized to a gate-level netlist using a logic synthesis tool and a description of the library. Finally, a place-and-route tool generates the physical design of the chip by first placing the library cells, and then routing them together with wires. The electrical properties of the wires are then extracted from this layout. Chip timing is then checked by calculating the critical path delays using these parasitics. Similarly power is estimated by taking the dot product of the wire capacitance vector and the toggle frequency vector.

There are three main problems with this approach:

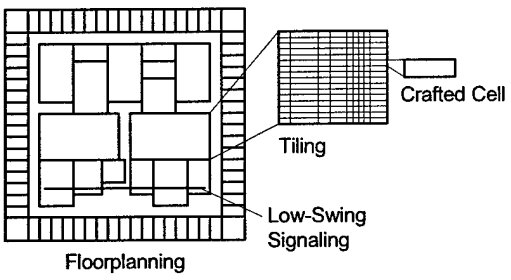
1. Little is known about the performance of the design until the very end. Power and timing are critically dependent on the resistance and capacitance of key wires which are only known after place and route.
2. The designer has little control. If the design fails to meet power and timing goals after P&R and after repowering the cells along the critical paths it is very difficult to fix the problem by changing the RTL. It is very frustrating when the solution (structuring the wiring) is obvious, but there is no way to tell the tools how to do it.
3. The result is unstable. If the designer changes the RTL late in the game to fix a logic bug, the resulting netlist and placement may change drastically creating a whole new set of timing and power problems.

To avoid these problems, the synthesis step is usually performed with a very conservative estimate of wiring parasitics (the wire model). The wire model picks a wire that is many sigma longer than the mean so that there are few surprises (wire loads larger than estimated) after place and route. Most gates are oversized for their wiring and most paths are optimized assuming pessimistic wire loads. The net result is logic that is much larger, much higher power, and slower than it needs to be.

2.2 There is a continuum of custom design approaches that structure the wiring

Custom design structures the wiring so that the parameters of key wires are known at the start of the design process, rather than the end. This gives a stable design with early visibility to timing and power problems. Moreover the designer has complete control over the result and can exploit the structure they see in the logic.

Figure 2: Custom Design Techniques



A number of different techniques for custom design that structure wires at different levels with different amounts of design effort are illustrated in Figure 2. They form a step-by-step approach to building better ASICs with less effort. The first several steps involve no custom cells, and only the last few steps involve custom circuits.

The first step is to floorplan the chip into small regions (usually less than 1000χ on a side, where χ is the minimum metal pitch) and impose a discipline on signals that cross between these regions (e.g., no combinational paths across regions). This floorplanning structures all of the inter-region wiring. The length and density of these wires is known up front and the logic and circuits can be built using this knowledge. For example, pipeline stages may be added to account for the known delay of long wires. Often this knowledge is exploited in the physical design by prerouting these signals.

The next step in structuring wires is to tile the *datapaths*. The designer partitions structured arrays into rows and columns and assigns library cells to each datapath cell. This is, in effect, floorplanning to a very fine granularity. A typical datapath cell is 1χ in the bit direction by 16-100χ in the function direction. In addition to structuring processor datapaths involving registers, adders, ALUs, and multiplexers, we have also found this approach useful in structuring arrays that exist in control logic for issue, decoding, scheduling, and matching. There are few large logic circuits that are devoid of structure.

At this point the designer becomes frustrated that it takes six library cells to make a register cell and realizes considerable savings by adding some new cells to the library. These *crafted cells* are simply large standard cells with functionality matched to the application. Their aspect ratio, pin placement, and electrical characteristics are all compatible with the standard-cell approach. These cells have three advantages. First by keeping all internal wiring short, global wiring is reduced and very small devices can be used to drive internal nets. Second, by folding together several functions, redundant 'safety' circuits (e.g., the inverters found on many standard-cell inputs and outputs) can be eliminated. Finally, judicious use of aggressive sizing and custom circuits can be employed in these cells. Crafted cells can be realized at different levels of ambition. The simplest crafted cells are just several library cells fused together, realizing just the first advantage. At the next level, arbitrary static CMOS circuits are employed within the cell. Finally, for very special cases, custom circuits are employed within the cell.

Our experience shows that the level of effort to develop a crafted cell, including generating all of the 'views' needed to support the standard-cell tools, is quite low (0.5-4 person days depending on complexity)[1][6] and that a very small number of cells (10-20) is sufficient to realize most datapaths. Moreover these cells are reusable. They really should already be in the library.

An orthogonal step, that can be applied independent of tiling or crafted cells, is to use signaling circuits to improve the performance of global signals. Structuring the global wiring at the floorplanning step, the electrical characteristics of global wires, resistance, capacitance, and coupling, become well characterized and well controlled. We can exploit this degree of control by using circuits that are optimized for the wires. For example, on recent chips we have employed low-swing (100mV) drivers with clocked receivers that cut wire latency by a factor of 3, increase repeater spacing by a factor of 3, and cut power by a factor of 10. The drivers and receivers fit into the standard-cell methodology, but can

only be employed on pre-routed wires that have well controlled characteristics and are shielded from full-swing CMOS signals. This approach would never work on the 'spaghetti' wiring generated by a place and route system.

The final step along our path is to do a *full custom* design. In a full custom design, there is aggressive use of special circuits and every device is sized for its load. Devices need not be organized into cells with 'on-grid' I/Os and 'safe' electrical interfaces. In short, there are no constraints. While full custom design does give the best absolute performance, it also requires tremendous effort and gives limited incremental returns compared to a crafted-cell approach with optimized global signaling. It is almost never justified.

3 Custom design structures the wiring which in turn enables optimized circuits.

In custom designs, effective partitioning of logic into primitives and the selection of optimal circuit styles is enabled by the synergy of pre-defined structure; explicit global and local wire planning; and detailed bit-slice floorplanning. Four specific benefits combine to give power savings and reduced delay. First, designers can size devices to match the actual load instead of relying on oversized devices configured for a generic load. Second, designers can preserve datapath regularity minimizing required area and eliminating unnecessary gate and wire loading. Third, designers can use the most appropriate logic style to implement a given function, for example a simple PLA instead of gates for a state machine. Lastly, the pre-planning and up-front design partitioning inherent in a custom approach enables designers to identify and apply fast circuits (domino, DCVSL, reduced swing, etc...) exactly where they will have the most impact.

3.1 Area and power are saved by using minimum sized devices (W=4λ devices rather than W=50λ devices)

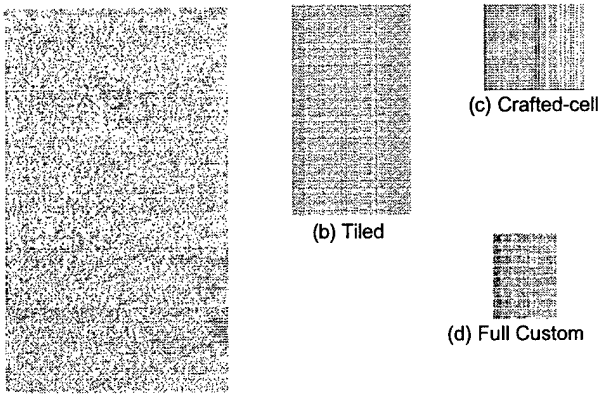
The smallest inverter in a typical 0.18um standard-cell library is 5µm/2.5µm (a 5µm PFET over a 2.5µm NFET) or about 50λ/25λ [5] where λ is half the minimum gate length. In crafted cells, where the wire loads are small and certain, we often employ a true minimum-sized inverter, 6λ/4λ. This has *one tenth* the total gate width of the smallest standard cell inverter and hence *one tenth* the input capacitance (3ff vs 30ff). As a result, where we can structure the wiring to keep wire loads small and well known these circuits require significantly less power and area and often result in smaller delays as well. Often a few crafted cells with small devices, iterated many times, account for a large fraction of the area and power of a chip even though most of the logical complexity of the chip is realized as automated layout with large devices.

3.2 Datapaths stay bit aligned reducing power, area, and load on critical circuits.

A cell-based methodology can be made to approach the quality of custom design by selectively adding a few *crafted cells* and manually tiling bit-slices. The regular layout of a datapath with bits aligned in one direction and functions aligned in the other results in extremely efficient wiring. The length of the bit lines and word (function control) lines are minimized as they can follow a straight path rather than wandering from side-to-side. Also, most of the wiring within a function remains local to a *cell* of the datapath resulting in short wires with small loads. Unfortunately, existing CAD tools are poor at identifying and extracting the bit-slice structure of datapaths. The space of possible placements is too large and the space of good solutions too small to find this structure using search. However, the advantage of datapath structure can be realized in an ASIC design flow by manually *tiling* logic into the cells of a datapath.

Datapath tiling can be exploited at several levels of customization. The impact on area and delay of different design styles is shown in Figure 3 and Table 1. These results are from four implementations of a 64-bit microprocessor register fetch stage [1][6]. This datapath includes a 7-ported 75 entry 64-bit register file, a six-entry reservation station, bypass multiplexors, 18-bit immediate insertion logic and thirteen 1-bit condition code registers. The baseline custom design is implemented in IBM's CMOS5L (Leff = 0.5µm, M2/M3 pitch 1.8µm) and contains 68361 transistors occupying 1864.8µm x 1344.6µm (1036χ x 747χ) area. All four designs in Figure 3 are drawn at the same scale.

Figure 3 Layout of a 64-bit datapath in four design styles



The custom approach (Figure 3d) was created with manually generated schematics, required 80 unique cells and each cell was manually placed and hand routed. The crafted-cell implementation (Figure 3c) was created from a structural verilog model which was then manually mapped to a basic 91 gate standard-cell library supplemented by 7 crafted cells. Bit-slices were manually tiled and function blocks are manually ordered. The design was automatically routed after the detailed placement. The 'bitsliced' standard cell implementation (Figure 3b) was created by synthesis of a *one bit slice* from the crafted-cell structural verilog source code targeting just the basic 91 cell library. The individual bit-slice was then placed and routed automatically. Finally, the bit-slices were ordered and assembled manually. The 'automated' standard cell implementation (Figure 3a) was created by synthesis *of the full design* from the crafted-cell structural verilog source code. The resulting netlist was then automatically placed and routed. The same metal layer conventions and external pin location constraints were used in all four cases. All four cases employed only static CMOS logic.

Table 1: Comparison of datapath design styles

	Custom	Crafted	Bitsliced Std Cells	Automated Std Cells
Area	1.0	1.64	5.25	14.50
Delay	1.0	1.11	2.23	
Gate Load	1.0	1.09	2.29	2.29
M2 Length	1.0	1.07	4.19	34.90
M3 Length	1.0	1.63	2.52	7.92

The differences in quality of the four designs highlights the benefits of both datapath structure and the use of design-specific custom cells. Structuring the design into bitslices reduced the area by almost a factor of 2.7 due to the regularity of the wiring. The next step, going to crafted cells saves another factor of 3.2. The crafted cells were able to overcome the inefficiency of building multiplexed input latches from the cells available in the library. The area advantage of full custom over the crafted cell approach was mainly due to the *gridding* penalty required to interoperate with the automated routing system.

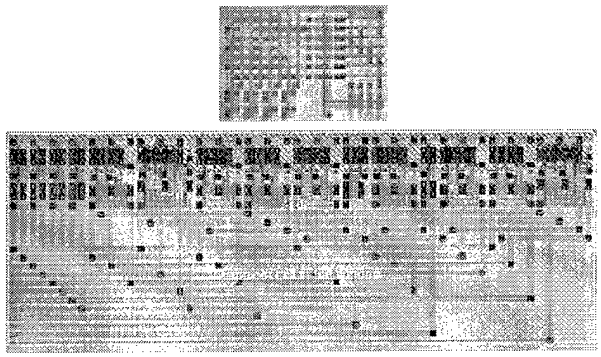
The results for delay are similar if a bit less dramatic. No delay results are available for the fully automated design. The crafted cell design outperforms the bitsliced design by almost exactly a factor of 2. This advantage is gained by removing redundant levels of logic and matching drivers to internal loads. The 1.1x speed advantage of full custom design is due to sub-optimal device sizing at the boundaries of crafted cells and increased wire parasitics due to the larger area.

The difference in metal length confirms the importance of preplanned structure. The custom design uses the minimal M2 & M3 resources as all contacts are made by abutment. While the crafted-cell design was autorouted, the manual placement guaranteed that all the routes were minimal. The increased metal usage for the bitsliced design is due to its larger area as well as the inefficiency that results from assembling complex cells from inexact primitives. The metal use in the fully automated design reflects the "spaghetti" wiring that results from fully automated place and route.

3.3 Structured circuits give much better density and performance.

Structured circuits, such as ROMs, RAMs, and PLAs organize the wires as a regular grid. The result is a considerably more compact layout. Also, because the characteristics of these wires are well controlled, structured circuits typically employ aggressive circuits and low-swing signaling to improve speed and power. A finite-state machine implemented as a well optimized PLA (or set of partitioned PLAs) requires about $4\chi^2$ of area per logic input. The circuit lays out in a regular grid, the product lines and output lines use a low voltage swing, and the flip-flops, with scan, are integrated into the output sense amp. As a result the area and power are an order of magnitude less than the corresponding gate circuit and the cycle time is much faster. The same FSM realized in gates takes about $100\chi^2$ per gate input not counting the flip-flops. As an illustration of the density of structured circuits, Figure 4 shows the same 4-input 4-output logic function implemented with standard cells and with a PLA. Despite the peripheral overhead of such a small design, the PLA implementation still occupies only about 1/8 the area as the standard cell design.

Figure 4: Comparison of standard-cell and PLA layout for a simple 4-input 4-output logic function



In reviewing a number of designs we have been amazed at the vast number of finite state controllers that should obviously be implemented as PLAs or microcode ROMs that are instead turned into a vast sea of gates because that is what the tools support. The designer writes her *microcode* in behavioral verilog, synthesizes it to a pile of gates, and then places and routes the circuit. What could have been a small, structured, stable, easy to modify PLA or ROM becomes a spaghetti connected set of standard cells that completely changes each time synthesis is rerun.

3.4 Structured layout enables the use of fast circuits

Fast circuits, such as domino logic, and low-swing high fan-in and fan-out circuits get much of the credit for the performance and power advantages of custom design over the standard-cell approach. In fact, these fast circuits are only possible because structured layout makes the wiring predictable, they cannot be applied independently. Also, dynamic circuits give less than a factor of 2 improvement over static CMOS [4]. Most of the improvement is due to the structure. Two key circuit concepts are widely used in custom designs: dynamic logic, and low-swing signals.

Dynamic logic is judiciously used in custom designs (full-custom or crafted cell) to reduce area and speed up critical paths. These circuits outperform static CMOS circuits for two reasons: (1) PFET loads and delays are largely eliminated from the critical path, and (2) the inputs switch at a lower point, V_{Tn}, than a static inverter, V_{inv}, and hence switch sooner. Dynamic circuits, however, are much more susceptible to noise and can only be used in environments where the coupling to nearby lines is well characterized. This coupling is easy to characterize with structured wiring and nearly impossible to do when the wiring is done last and randomly routed.

Low swing signals are used on high capacitance nets where the delay and power of amplifying the signal at the far end is less than the delay and power required to drive the signal through full-swing [2]. The most common application of this technique is the bit lines on a static RAM which swing only 100-200mV. However, it can also be applied to high fan-in circuits (like a match line in a CAM),

and high fan-out circuits (distributing a global 'stall' signal), and to circuits that have both high fan-in and fan-out (like a crossbar switch). The savings in area, delay, and power compared to a standard cell approach are considerable. In one switch that we realized, the design was simply not feasible using standard cells, but was fairly easy using structured design. The power savings are perhaps the most dramatic, by swinging through less than 10% of the power supply, these circuits dissipate less than 10% the power of the full-swing alternative. Like dynamic circuits, low-swing circuits cannot be applied independently. They depend on structured layout to give a well characterized wire to drive and to isolate the low-swing signal from noisy full swing signals. These circuits cannot be used with wiring that is routed last with coupling that is not well controlled.

4 CAD tools and IP offerings should embrace custom design

While we have had considerable success with the selective application of custom design techniques, many designers find it difficult to apply these techniques because they require a departure from the standard CAD flow. Also, they are not supported by almost all sources of reusable designs. Both of these problems could be easily remedied.

As designers, we would like to see CAD tools that embrace a selective custom approach to ASIC design. Specifically useful would be:

1. Tools that support datapath tiling. Starting with a verilog deck annotated with row and column positions and identifying signals as bit lines and word lines these tools would propagate this information to place the logic into specific bit cells and to route bit-lines and word-lines along single tracks.
2. Tools that streamline the verification, characterization, and integration of new cells. Designing the circuits and layout for a new crafted cell is not difficult. Most of the effort goes into validating this cell and generating the numerous *views* needed to integrate this cell into the standard CAD flow. This process could be completely automated.
3. Tools that easily support the manual placement of key signal groups, repeaters, and shielding during the floorplanning process and use this placement to guide logic placement. Such tools should also support adding noise and load constraints on select signals to facilitate the use of fast circuits.

Such tools would represent a shift on the part of the CAD vendors from a logic-centric view of design toward a wire-centered view of design. In the logic-centric view, the logic design comes first and is specified by the designer while the physical design (the wiring) comes last and is automatically generated. In the wire-centered view, the global wires come first, then the logic design and both are specified by the user. The tools then generate a detailed placement, implement the local wiring, and tidy up the loose ends. This arrangement gives the user control over the structure of the design.

The situation is similar with IP vendors. Most IP today is available as a synthesizable Verilog deck. When put through a standard ASIC flow, this generates layout with little structure. In some cases, IP vendors offer customized placement and even layout in a particular vendor's process, but such hard IP is not portable and hence not widely used.

What is needed is portable IP that captures structure and, in select cases, optimized circuits. Tools that support a wire-centered view of design would enable such portable, optimized IP. The IP vendor could encode the datapath structures within a design (even those in the 'control' blocks), pack circuitry into crafted cells, and even employ fast circuits, all in a portable manner. If IP blocks are to be widely used, they should be highly optimized. Providing IP as a Verilog deck permits only RTL optimization. Providing portable, structured IP permits optimization of circuits and layout as well.

We would like to see the IP industry provide ASIC-compatible, portable modules ranging from adders to microprocessors to network switches with the speed, area, and power of a full custom design.

5 Conclusion

Much of the gap between the performance of ASICs and the performance of full-custom ICs can be closed by selectively applying custom design techniques to structure ASICs. Floorplanning and datapath tiling to the level of bit cells fixes the characteristics of key wires early in the design process. This wires-first approach gives early visibility of timing and power problems and gives the designer control over the most important performance factor - wire loads. Building a small set of *crafted* datapath cells gives further benefits in area, delay, and power by keeping local wires short enabling the use of small drivers. Structuring the wiring through these techniques also enable the use of fast circuits by providing the controlled environment they require. As process geometries shrink, structured wiring will become even more critical to the performance of ASICs. CAD vendors and IP vendors should recognize this trend and provide tools and cores that embrace a wire-centered view of design.

References

[1] CHANG, ANDREW, *VLSI Datapath Choices: Cell-Based Versus Full-Custom*, SM Thesis, Massachusetts Institute of Technology, February 1998.

[2] DALLY, WILLIAM J., AND POULTON, JOHN W., *Digital Systems Engineering*, Cambridge University Press, 1998, Chapter 8.

[3] GRONOWSKI, PAUL E., BOWHILL, WILLIAM J., PRESTON, RONALD P., GOWAN, MICHAEL K., AND ALLMON, RANDY L., High-Perfomance Microprocessor Design, *IEEE Journal of Solid-state Circuits*. Vol 33. No 5., May 1998, pp. 676-686.

[4] HARRIS DAVID I., OBERMAN, STUART F., HOROWITZ, MARK A., "SRT Division Architectures and Implementations", *Proceedings of 13th IEEE International Symposium on Computer Arithmetic*, July 1997, pp 18-25.

[5] IBM CORPORATION, *SA-27E ASIC Databook*, February 2000.

[6] KECKLER, STEPHEN W., DALLY, WILLIAM J., CHANG, ANDREW., CARTER, NICHOLAS P., LEE, WHAY SING., "The MIT Multi-ALU Processor", *Hot Chips IX*, August 1997, pp 1-8.